Data Generation, Manipulation, and Conversion

/*Working with string data

CHAR Holds fixed-length, blank-padded strings. MySQL allows CHAR values up to 255

VARCHAR Holds variable-length strings. MySQL permits up to 65,535 characters in a varchar column

TEXT MySQL has multiple text types (tinytext, text, mediumtext, and longtext) for documents up to 4 GB in size*/

```
USE favorite_food_details;
```

-- String Generation

```
CREATE TABLE string_tbl (
    char_field CHAR(30),
    vchar_field VARCHAR(30),
    text_field TEXT
);

INSERT INTO string_tbl (char_field, vchar_field, text_field)
VALUES ('This is char data', 'This is varchar data', 'This is text data');

UPDATE string_tbl
SET
    vchar_field = 'This is varchar data but iam exceding the length to show the error';

SELECT @@session.sql_mode; -- to check which mode you are using in
```

-- Including Single Quotes

```
UPDATE string_tbl
SET text_field = 'This string doesn't work'';

UPDATE string_tbl
SET
    text_field = 'This string didn''t work, but it does now';

SELECT
    *
FROM
    string_tbl;

-- quote() function

SELECT
    text_field
FROM
    string_tbl;

SELECT
    quote(text_field)
FROM
    string_tbl;
```

/*Including special characters - When working with the French and German languages, for example, you might need to include accented characters such as é and ö. The SQL Server and MySQL servers include the built-in function char() so that you can build strings from any of the 255 characters in the ASCII character set*/

```
SELECT 'abcdefgh', CHAR(97, 98, 99, 100, 101, 102, 103);
```

/*Thus, the 97th character in the ASCII character set is the letter a.*/

```
SELECT CHAR(128,129,130,131,132,133,134,135,136,137);
```

```
SELECT CHAR(158,159,160,161,162,163,164,165);
```

-- you will need to familiarize yourself with the layout of your character set to locate specific characters

/*Building strings character by character can be quite tedious, especially if only a few of the characters in the string are accented. Fortunately, you can use the **concat()** function to concatenate individual strings, some of which you can type while others you can generate via the char() function. For example, the following shows how to build the phrase danke schön using the concat() and char() functions:*/

```
SELECT CONCAT('danke sch', CHAR(148), 'n');
```

/*If you have a character and need to find its ASCII equivalent, you can use the ascii() function*/

```
SELECT ASCII('ö');
```

```
SELECT ASCII('Pradeepchandra'); -- takes the leftmost character in the string and returns a number
```

/*String Manipulation

We explore two types of string functions: those that return numbers and those that return strings.

Length of the string - number of characters*/

```
SELECT
    LENGTH(char_field) AS car_length,
    LENGTH(vchar_field) AS varchar_length,
    LENGTH(text_field) AS text_length
FROM
    string_tbl;
```

/*The MySQL server removes trailing spaces from char data when it is retrieved*/

/*POSITION function finds the position of the characters you provide*/

```
SELECT
    POSITION('data' IN vchar_field)
FROM
    string_tbl;
```

/*If you want to start your search at something other than the first character of your target string, you will need to use **the locate()** function, which is similar to the position() function except that it allows an optional third parameter, which is used to define the search's start position.*/

```
SELECT
    LOCATE('data', vchar_field, 5)
FROM
    string_tbl;
```

/*String Comparisions

Another function that takes strings as arguments and returns numbers is the string comparison function strcmp().

If string1 = string2, this function returns 0

If string1 < string2, this function returns -1

If string1 > string2, this function returns 1*/

```
DELETE FROM string_tbl;

SELECT
    *
FROM
    string_tbl;

INSERT INTO string_tbl(vchar_field)
VALUES ('abcd'),
('xyz'),
('QRSTUV'),
('qrstuv'),
('12345');
```

```
SELECT
    vchar_field
FROM
    string_tbl
ORDER BY vchar_field;


SELECT
    STRCMP('12345', '12345') AS 12345_12345,
    STRCMP('abcd', 'xyz') AS abcd_xyz,
    STRCMP('abcd', 'QRSTUV') AS abcd_QRSTUV,
    STRCMP('qrstuv', 'QRSTUV') AS qrstuv_QRSTUV,
    STRCMP('12345', 'xyz') AS 12345_xyz,
    STRCMP('xyz', 'qrstuv') AS xyz_qrstuv;


/*MySQL’s strcmp() function is case insensitive, which is something
to remember when using the function*/


/*Along with the strcmp() function, MySQL also allows you to use the
like and regexp operators to compare strings in the select clause.
Such comparisons will yield 1 (for true) or 0 (for false).*/
use sakila;


SELECT
    name, name LIKE '%y' AS ends_in_y
FROM
    category;


SELECT
    name, name LIKE 'a%' AS starts_with_a
FROM
    category;
```

/*String functions that return strings*/

```
use favorite_food_details;

DELETE FROM string_tbl;

INSERT INTO string_tbl (text_field)
VALUES ('This string was 29 characters');

UPDATE string_tbl
SET text_field = CONCAT(text_field, ', but now it is longer');

SELECT
    text_field
FROM
    string_tbl;
```

```
/*Another common use for the concat() function is to build a string
from individual pieces of data. For example, the following query
generates a narrative string for each customer:*/


use sakila;


SELECT
    CONCAT(first_name,
            ' ',
            last_name,
            'has been a customer since ',
            DATE(create_date)) AS customer_details
FROM
    customer;


/*MySQL includes the insert() function, which takes four arguments:
the original string, the position at which to start, the number of
characters to replace, and the replacement string. Depending on the
value of the third argument, the function may be used to either insert
or replace characters in a string.*/


SELECT INSERT('goodbye world', 9, 0, 'cruel ') AS insert_statement;


SELECT INSERT('Goodbye World', 1, 7, 'Hello') AS replaced;


/*The following example extracts five characters (SUBSTRTING) from
a string starting at the ninth position:*/


SELECT SUBSTRING('goodbye cruel world', 9, 11);
```

/*WORKING WITH NUMERIC DATA

You can type a number, retrieve it from another column, or generate it via a calculation.

All the usual arithmetic operators (+, -, *, /) are available for performing calculations, and parentheses may be used to dictate precedence, as in:*/

```
SELECT (45 * 88) / (78 - (41 + 15) + 12) AS Answer;
```

/*The **modulo operator**, which calculates the remainder when one number is divided into another number, is implemented in MySQL*/

```
SELECT MOD(10, 4) AS Remainder;
```

```
SELECT MOD(120.5, 4) AS Remainder_of_a_float;
```

/* **the pow()** function which returns one number raised to the power of a second number*/

```
SELECT POW(2, 8) AS power_;
```

```
SELECT
    POW(2, 10) AS kilobyte,
    POW(2, 20) AS megabyte,
    POW(2, 30) AS gigabyte,
    POW(2, 40) AS terabyte;
```

/* Controlling Number Precision - ceil(), floor(), round(), and truncate()

The ceil() and floor() functions are used to round either up or down to the closest integer*/

```
SELECT CEIL(72.445) AS ceil_value, FLOOR(72.445) AS floor_value;
```

/*Remember that ceil() will round up even if the decimal portion of a number is very small, and floor() will round down even if the decimal portion is quite significant*/

```
SELECT CEIL(72.000000001)AS ceil_value, FLOOR(72.999999999)AS floor_value;
```

/*If this is a bit too severe for your application, you can use the round() function to

round up or down from the midpoint between two integers*/

```
SELECT
    ROUND(72.49999) AS Round_Value,
    ROUND(72.5) AS Round_Value,
    ROUND(72.50001) AS Round_Value;
```

/*The round() function allows an optional second argument to specify how many digits to the right of the decimal place to round to.*/

```
SELECT
    ROUND(72.49999, 2) AS Round_Value,
    ROUND(72.5, 3) AS Round_Value,
    ROUND(72.50001, 1) AS Round_Value;
```

```
/*the truncate() function allows an optional second argument to specify the number of digits to the right of the decimal, but truncate() simply discards the unwanted digits without rounding.*/


SELECT
    TRUNCATE(72.0909, 1),
    TRUNCATE(72.0909, 2),
    TRUNCATE(72.0909, 3);
```

/*Handling Signed Data*/

```
/*
+------------+--------------+---------+
| account_id |    acct_type | balance |
+------------+--------------+---------+
| 123        | MONEY MARKET |  785.22 |
| 456        | SAVINGS      | 0.00    |
| 789        | CHECKING     | -324.22 |
+------------+--------------+---------+


SELECT account_id, SIGN(balance), ABS(balance)
FROM account;
+------------+---------------+--------------+
| account_id | SIGN(balance) | ABS(balance) |
+------------+---------------+--------------+
| 123        | 1             | 785.22       |
| 456        | 0             | 0.00         |
| 789        | -1            | 324.22       |
+------------+---------------+--------------+
```

The second column uses the **sign()** function to return –1 if the account balance is negative, 0 if the account balance is zero, and 1 if the account balance is positive. The third column returns the absolute value of the account balance via the **abs()** function.*/